MaRVIN: A platform for large-scale analysis of Semantic Web data

Oren, E.; Kotoulas, S.; Anadiotis, G.; Siebes, R.M.; ten Teije, A.C.M.; van Harmelen, F.A.H.

published in
Proceedings of the WebSci09: Society On-Line
2009

document version
Early version, also known as pre-print

MARVIN: A platform for large-scale analysis of Semantic Web data

Eyal Oren, Spyros Kotoulas, George Anadiotis, Ronny Siebes, Annette ten Teije, and Frank van Harmelen
Department of Computer Science, Vrije Universiteit Amsterdam, the Netherlands

***Abstract*—Web Science requires efficient techniques for analysing large datasets. Many Semantic Web problems are difficult to solve through common divide-and-conquer strategies, since they are hard to partition. We present MARVIN, a parallel and distributed platform for processing large amounts of RDF data, on a network of loosely-coupled peers. We present our *divide-conquer-swap* strategy and show that this model converges towards completeness. We evaluate performance, scalability, load balancing and efficiency of our system.**

I. ANALYSING WEB DATA

Web Science involves, amongst others, the analysis and interpretation of data and phenomena on the Web [9]. Since the datasets involved are typically very large, efficient techniques are needed for scalable execution of analysis jobs over these datasets.

Traditionally, scaling computation through a divide-and-conquer strategy has been successful in a wide range of data analysis settings. Dedicated techniques have been developed for analysis of Web-scale data through a divide-and-conquer strategy, such as MapReduce [5].

Over the recent years, large volumes of Semantic Web data have become available, to the extent that the data is quickly outgrowing the capacity of storage systems and reasoning engines. Through the "linking open data" initiative, and through crawling and indexing infrastructures [13], datasets with millions or billions of triples are now readily available. These datasets contain RDF triples and many RDFS and OWL statements with implicit semantics [6].

From a Web Science viewpoint, these datasets are often more interesting than the Web graph [9] of page hyperlinks. First, since these datasets contain typed relations with particular meaning, they can be subjected to more detailed analysis. Secondly, most of these datasets are not annotated Web pages but rather interlinked exports of the "deep Web", which has traditionally been hard to obtain and analyse [14].

However, to process, analyse, and interpret such datasets collected from the Web, infrastructure is needed that can scale to these sizes, and can exploit the semantics in these datasets. In contrast to other analysis tasks concerning Web data, it is not clear how to solve many Semantic Web problems through divide-and-conquer, since it is hard to split the problem into independent partitions.

To illustrate this problem we will focus on a common and typical problem: computing the deductive closure of these datasets through logical reasoning. Recent benchmarks [2, 8] show that current RDF stores can barely scale to the current volumes of data, even without this kind of logical reasoning.

II. SCALABLE RDF REASONING

To deal with massive volumes of Semantic Web data, we aim at building RDF engines that offer *massively scalable* reasoning. In our opinion, such scalability can be achieved by combining the following approaches:

- using *parallel hardware* which runs *distributed algorithms* that exploit such hardware regardless of the scale, varying from tens of processors to many hundreds (as in our experiments) or even many thousands.
- designing *anytime algorithms* that produce *sound results* where the degree of completeness increases over time. Such algorithms can trade the speed with which the inference process converges to completeness against the size of the dataset, while still guaranteeing eventual completeness.
- our novel *divide-conquer-swap* strategy, which extends the traditional approach of divide-and-conquer with an iterative procedure whose result converge towards completeness over time.

We have implemented our approach in MARVIN[1], a parallel and distributed platform for processing large amounts of RDF data. MARVIN consists of a network of loosely-coupled machines using a peer-to-peer model and does not require splitting the problem in independent subparts. MARVIN is based on the approach of *divide-conquer-swap*: peers autonomously partition the problem in some manner, each operate on some subproblem to find partial solutions, and then re-partition their part and swap it with another peer; all peers keep re-partitioning, solving, and swapping to find all solutions. We show that this model is sound, converges and reaches completeness eventually.

III. RELATED WORK

Several techniques for distributed reasoning are based on distributed hashtables (DHTs) [11]. Cai and Frank [4] introduce a basic schema for indexing RDF in DHTs. This layout leads to uneven load distribution between nodes since term popularity in RDF exhibits a power-law distribution [13]. Fang *et al.* [7] have an iterative forward-chaining procedure similar to ours but do not address load-balancing issues. Kaoudi *et al.* [10] propose a backward-chaining algorithm which seems promising, but no conclusions can be drawn given the small

[1]Named after Marvin, the paranoid android from the Hitchhiker's Guide to the Galaxy. Marvin has "a brain the size of a planet", which he can seldomly use: the true horror of Marvin's existence is that no task would occupy even the tiniest fraction of his vast intellect.

dataset (10^4 triples) and atypical evaluation queries. Battré *et al.* [1] perform limited reasoning over the locally stored triples and introduce a policy to deal with load-balancing issues, but only compute a fraction of the complete closure.

Serafini and Tamilin [16] perform distributed description logics reasoning; the system relies on manually created ontology mappings, which is quite a limiting assumption, and its performance is not evaluated. Schlicht and Stuckenschmidt [15] distribute the reasoning rules instead of the data: each node is only responsible for performing a specific part of the reasoning process. Although efficient by preventing duplicate work, the weakest node in this setup becomes an immediate bottleneck and a single-point-of-failure, since all data has to pass all nodes for the system to function properly.

IV. Our approach: Divide-conquer-swap

MARVIN operates using the following "main loop", run on a grid of compute nodes; in this loop, steps 3–5 are repeated infinitely, and operational statistics are continously gathered from all compute nodes:

Algorithm 1 Divide-conquer-swap

1) The input data is divided into smaller chunks, which are stored on a shared location.
2) A large number of "data processors" is started on the nodes of the grid (the nature of these processors depends on the task at hand, these could be for example be reasoners or social graph analysers).
3) Each node reads some input chunks and computes the corresponding output of this input data at its own speed.
4) On completion, each node selects some parts of the computed data, and sends it to some other node(s) for further processing. Asynchronous queues are used to avoid blocking communication.
5) Each node copies (parts of) the computed data to some external storage where the data can be queried on behalf of end-users. These results grow gradually over time, producing anytime behaviour.

When performing *divide-conquer-swap*, we have to address two key trade-offs: on the one hand, we want to solve the problem as efficiently as possible, while on the other hand we want to minimise communication overhead and ensure that the processing load is shared equally over all nodes. Secondly, to maximise efficiency with minimal communication overhead, we might let nodes process partially overlapping partitions; however, we need an efficient method to detect when nodes produce duplicate data, to prevent unneccesary computations. We will discuss our approach to each of these trade-offs in turn.

A. Load balancing and efficient computation

All communication in MARVIN is *pull-based*: peers explicitly *request* data from other peers, which prevents overloading peers with too much data. Nodes also protect themselves against high loads by ignoring incoming requests for data when they have more than some threshold of their communication channels in use. Above some other threshold, nodes will stop all reasoning and reject incoming messages until they empty their communication queues. All communication is sanity-checked using timeouts: messages are dropped if they cannot be delivered within a certain timeframe.

Reasoning in a distributed system involves a trade-off between overall efficiency and individual load-balancing: an efficient distribution would route all triples involving some given term to one node, where all inferences based on these triples can be drawn. Since however terms are distributed very unevenly (some terms are much more popular than others - see [13]), the nodes responsible for these terms will have a much higher load.

As discussed in section III, existing approaches that use distributed hash-tables for RDF reasoning suffer from such load-balancing problems. A uniform *random* distribution of triples over nodes solves such load-balancing problems (since all are distributed evenly) but is not very efficient for drawing inferences.

We have implemented a hybrid approach called *pull-DHT*, with two characteristic features:

- in contrast to common DHT-based approaches, nodes pull data instead of getting triples pushed to them, and
- when asking peers for data, nodes still get a random distribution of triples, but instead of uniform, the distribution is biased towards triples that fall into their address space (based on the hash value of the triple and the node's rank in the network).

The *pull-DHT* strikes a balance between the perfectly balanced but inefficient random routing and the efficient but unbalanced DHT routing.

B. Duplicate detection and removal

Since our aim is to minimise the time spent for deduction of the closure, we should spend most time computing new facts instead of re-computing known facts. Duplicate triples can be generated for several reasons like redundancy in the initial dataset, sending identical triples to several peers or deriving the same conclusions from different premises.

In reasonable quantities duplicate triples may be useful: they may participate, in parallel, in different deductions. In excess, however, they pose a major overhead: they cost time to produce and process and they occupy memory and bandwidth. Therefore, we typically want to limit the amount of duplicate triples in the system.

To remove duplicates from the system, they need to be detected. However, given the size of the data, peers cannot keep a list of *all* previously seen triples in memory: even using an optimal data structure such as a Bloom filter [3] with only 99% confidence, storing the existence of 8 billion triples would occupy some 9.5GB of memory on each peer.

We tackle this issue by distributing the duplicate detection effort, implementing a *one-exit door policy*: we assign the responsibility to detect each triple's uniqueness to a single peer, using a uniform hash function: $exit\_door(t) = hash(t) \bmod N$, where t is a triple and N is the number of nodes. The

exit door uses a bloomfilter to detect previously encountered triples: it marks the first copy of each triple as *master copy*, and removes all other subsequent copies.

For large number of nodes however, the *one-exit door policy* becomes less efficient since the probability of a triple to randomly appear at its exit door is $\frac{1}{N}$ for N number of nodes. Therefore, we have an additional and configurable *sub-exit door policy*, where some k peers are responsible for explicitly routing some triples to an exit door, instead of waiting until the triples arrive at the designated exit door randomly.

A final optimisation that we call the *dynamic sub-exit door policy* makes k dependent on the number of triples in each local output buffer - raising k when the system is loaded and lowering it when the system is underutilized. This mechanism effectively works as a pressure valve, relieving the system when pressure gets too high. This policy is implemented with two thresholds: if the number of triples in the *output pool* exceeds t_{upper} then we set $k = N$, if it is below t_{lower} then we set $k = 0$.

V. Eventual completeness

In this section we will provide a qualitative model to study the completeness of MARVIN. Assuming a sound external procedure in the "conquer" step, overall soundness is evident through inspection of the basic loop, and we will not discuss it further.

The interesting question is not only *whether* MARVIN is complete: we want to know *to which extent* it is complete, and how this completeness *evolves over time*. For such questions, tools from logic do not suffice since they treat completeness as a binary property, do not analyse the degree of completeness and do not provide any progressive notion of the inference process. Instead, an elementary statistical approach yields more insight.

Let C^* denote the deductive closure of the input data: all triples that can be derived from the input data. Given MARVIN's soundness, we can consider each inference as a "draw" from this closure C^*. Since MARVIN derives its conclusions gradually over time, we can regard MARVIN as performing a series of repeated draws from C^* over time. The repeated draws from C^* may yield triples that have been drawn before: peers could re-derive duplicate conclusions that had been previously derived by others. Still, by drawing at each timepoint t a subset $C(t)$ from C^*, we gradually obtain more and more elements from C^*.

In this light, our completeness question can be rephrased as follows: how does the union of all sets $C(t)$ grow with t? Will $\cup_t C(t) = C^*$ for some value of t?

At which rate will this convergence happen? Elementary statistics tells us that if we draw t times a set of k elements from a set of size N, the number of distinct drawn elements is expected to be $N \times (1 - (1 - k/N)^t)$. Of course, this is the *expected* number of distinct drawn elements after t iterations, since the number of drawn duplicates is governed by chance, but the "most likely" (expected) number of distinct elements after t iterations is $N \times (1-(1-k/N)^t)$, and in fact the variance of this expectation is very low when k is small compared to N.

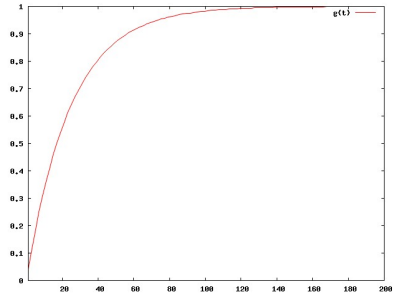

Fig. 1. Predicted rate of unique triples produced

In our case, $N = |C^*|$, the size of the full closure, and $k = |C(t)|$, the number of triples jointly derived by all nodes at time t, so that the expected completeness $\gamma(t)$ after t iterations is:

$$\gamma(t) = (1 - (1 - \frac{|C(t)|}{|C^*|})^t)$$

Notice that the boundary conditions on $\gamma(t)$ are reasonable: at $t = 0$, when no inference has been done, we have maximal incompleteness ($\gamma(0) = 0$); for trivial problems where the peers can compute the full closure in a single step (i.e. $|C(1)| = |C^*|$), we have immediate full completeness ($\gamma(1) = 1$); and in general if the peers are more efficient (ie they compute a larger slice of the closure at each iteration), then $|C(t)|/|C^*|$ is closer to 1, and $\gamma(t)$ converges faster to 1, as expected. The graph of unique triple produced over time, as predicted by this model, is shown in figure 1. The predicted completeness rate fits the curves that we find in experimental settings, shown in the next section.

This completeness result is quite robust. In many realistic situations, at each timepoint the joint nodes will only compute a small fraction of the full closure ($C(t) \ll |C^*|$), so that $\gamma(t)$ is a reliable expectation with only small variance. Furthermore, completeness still holds when $|C(t)|$ decreases over t, which would correspond to the peers becoming less efficient over time, through for example network congestion or increased redundancy between repeated computations.

Our analytical evaluation shows that *reasoning in* MARVIN *converges and reaches completeness eventually*. Still, convergence time depends on system parameters such as the size of internal buffers, the routing policy, and the exit policy. In the next section, we report on empirical evaluations to understand the influence of these paramaters.

VI. Evaluation

We have implemented MARVIN in Java, on top of Ibis, a high-performance communication middleware [12]. Ibis offers an integrated solution that *transparently* deals with many complexities in distributed programming such as network connectivity, hardware heterogeneity, and application deployment.

We have experimented with many internal parameters such as data distribution or routing policy; MARVIN is equipped with tools for logging key performance indicators, to facilitate experimentation until optimal settings for the task at hand are found.

Experiments were run on the Distributed ASCI Supercomputer 3 (DAS-3), a five-cluster grid system, consisting in total of 271 machines with 791 cores at 2.4Ghz, with 4Gb of RAM per machine. All experiments used the Sesame in-memory store with a forward-chaining RDFS reasoner. All experiments we limited to a max. runtime of one hour, and were run on smaller parts of the DAS-3, as detailed in each experiment.

The datasets used were RDF Wordnet[2] and SwetoDBLP[3]. Wordnet contains around 1.9M triples, with 41 distinct predicates and 22 distinct classes; the DBLP dataset contains around 14.9M triples, with 145 distinct predicates and 11 distinct classes. Although the schemas used are quite small, we did not take exploit this fact in our algorithm (eg. by distributing the schemas to all nodes a priori) because such optimisation would not be possible for larger or initially unknown schemas.

A. Baseline: null reasoner

To validate the behavior of the baseline system components such as buffers and routing algorithms, we created a "null reasoner" which simple outputs all its input data. We thus measure the throughput of the communication substrate and the overhead of the platform.

In this setup, the system reached a sustained throughput of 72.9 Ktps (thousand triples per second), with a sustained transfer rate of 26.5 MB/s per node. Typically, just indexing RDF data is slower (some 20–40 Ktps) [2], and reasoning is even more computationally expensive. Therefore, we can expect the inter-node communication (in the network used during our experiments) not to be a performance bottleneck.

B. Scalability

We have designed the system in order to scale to a large number of nodes. The Ibis middleware is based on solid grid technology which allows MARVIN to scale to a large number of nodes. Figure 2 shows the speedup gained by additional computational resources (using random routing, on the SwetoDBLP dataset), showing the number of unique triples produced for a system of 1–64 nodes. As we can see, the system scales gracefully.

The sharp bends in the growth curves (especially with a small number of nodes) are attributed to the dynamic exit doors opening: having reached the t_{upper} threshold, the nodes start sending their triples to the exit door, where they are counted and copied to the storage bin.

nodes	time (min)	speedup	scaled speedup
1	44	–	–
2	30	1.47	0.73
4	26	1.69	0.42
8	20	2.20	0.28
16	9.5	4.63	0.29
32	6.2	7.10	0.22
64	3.4	12.94	0.20

TABLE I
ABSOLUTE AND SCALED SPEEDUP FOR SWETODBLP DATASET

[2]http://larkc.eu/marvin/experiments/wordnet.nt.gz
[3]http://larkc.eu/marvin/experiments/swetodblp.nt.gz

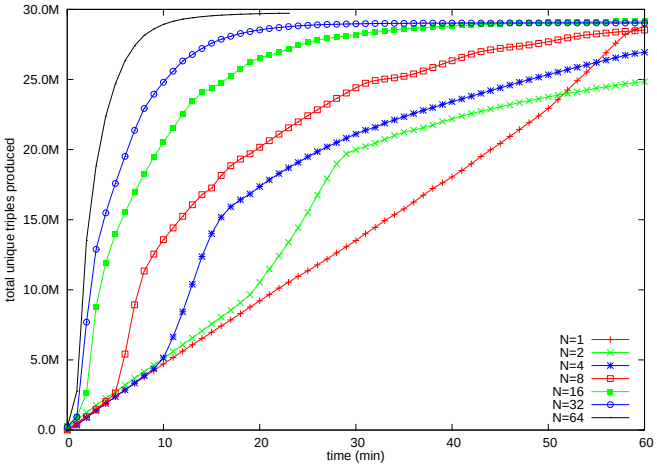

Fig. 2. Triples derived using an increasing number of nodes

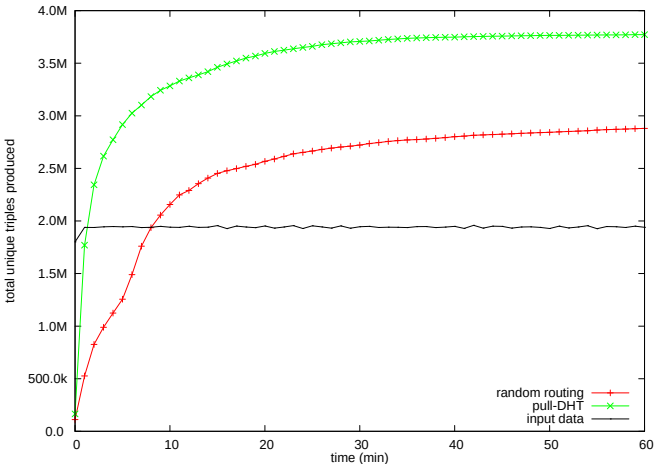

Fig. 3. Triples derived using different routing strategies

Table I shows the time needed to produce some fixed number of triples in the SwetoDBLP dataset (namely, 20M triples). It shows the amount of time needed over different numbers of nodes, and compute the corresponding speedup (total time spent compared to time spent on single node) and the scaled speedup (speedup divided by number of nodes). A perfect linear speedup would equal the number of nodes and result in a scaled speedup (speedup divided by number of nodes) of 1. To the best of our knowledge no relevant literature is available in the field to compare these results, but a sublinear speedup is to be expected in general.

C. Load balancing and efficiency

Figure 3 shows a comparison of the *random triple routing* with the *pull-DHT*. The graphs show the time needed for producing the number of unique triples shown. The *pull-DHT* outperforms the *random routing*, converging faster and producing more unique triples in total. In all experiments presented in this section, the load was balanced evenly over all nodes.

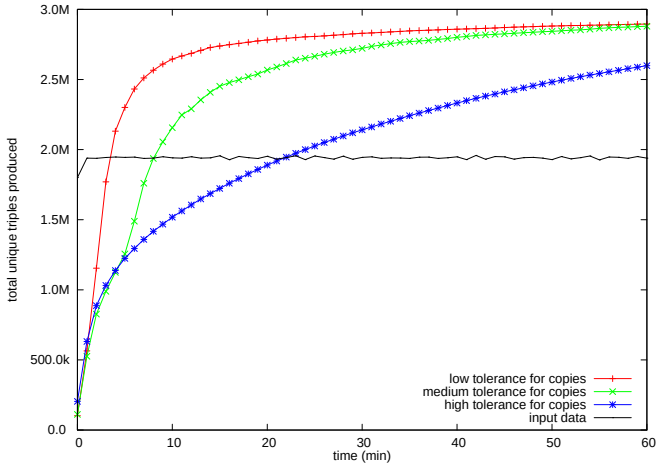

Fig. 4. Triples derived using the dynamic exit-door policy

D. Duplicate detection and removal

We have experimented with three different settings of the *dynamic sub-exit door*: "low" where $t_{lower} = \alpha, t_{upper} = 2\alpha$; "medium" where $t_{lower} = 2\alpha, t_{upper} = 4\alpha$; "high" where $t_{lower} = 4\alpha, t_{upper} = 8\alpha$, where α is the number of input triples / N.

These different settings were tested on the Wordnet dataset, using 16 nodes with the *random* routing policy. The results are shown in figure 4. As we can see, in the "low" setting, the system benefits from having low tolerance to duplicates: they are removed immediately, leaving bandwidth and computational resources to produce useful unique new triples. On the other hand, the duplicate detection comes at the cost of additional communication needed to send triples to the exit doors (not shown in the figure).

VII. Conclusion

We have presented a platform for analysing Web data, with a focus on the Semantic Web. To process and interpret these datasets, we need an infrastructure that can scale to Web size and exploit the available semantics. In this paper, we have focused on one particular problem: computing the deductive closure of a dataset through logical reasoning.

We have introduced MARVIN, a platform for massive distributed RDF inference. MARVIN uses a peer-to-peer architecture to achieve massive scalability by adding computational resources through our novel *divide-conquer-swap* approach. MARVIN guarantees eventual completeness of the inference process and produces its results gradually (anytime behaviour). Through its modular design and its built-in instrumentation, MARVIN provides a versatile experimentation platform with many configurations.

We have experimented with various reasoning strategies using MARVIN. The experiments presented show that MARVIN scales gracefully with the number of nodes, that the communication overhead is not the bottleneck during computation, and that duplicate detection and removal is crucial for performance. Furthermore, we have introduced an initial *pull-DHT* routing policy, improving performance without disturbing load balancing.

Acknowledgements: This work is supported by the European Commission under the LarKC project (FP7-215535).

References

[1] D. Battré, A. Höing, F. Heine, and O. Kao. On triple dissemination, forward-chaining, and load balancing in DHT based RDF stores. In *Proceedings of the VLDB Workshop on Databases, Information Systems and Peer-to-Peer Computing (DBISP2P)*. 2006.

[2] C. Bizer and A. Schultz. Benchmarking the performance of storage systems that expose SPARQL endpoints. In *Proceedings of the ISWC Workshop on Scalable Semantic Web Knowledge-base systems*. 2008.

[3] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[4] M. Cai and M. Frank. RDFPeers: A scalable distributed RDF repository based on a structured peer-to-peer network. In *Proceedings of the International World-Wide Web Conference*. 2004.

[5] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the USENIX Symposium on Operating Systems Design & Implementation (OSDI)*, pp. 137–147. 2004.

[6] M. Dean. Towards a science of knowledge base performance analysis. In *Proceedings of the ISWC Workshop on Scalable Semantic Web Knowledge-base systems*. 2008.

[7] Q. Fang, Y. Zhao, G. Yang, and W. Zheng. Scalable distributed ontology reasoning using DHT-based partitioning. In *Proceedings of the Asian Semantic Web Conference (ASWC)*. 2008.

[8] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3:158–182, 2005.

[9] J. Hendler, N. Shadbolt, W. Hall, T. Berners-Lee, *et al.* Web science: An interdisciplinary approach to understanding the web. *Communications of the ACM*, 51, 2008.

[10] Z. Kaoudi, I. Miliaraki, and M. Koubarakis. RDFS reasoning and query answering on top of DHTs. In *Proceedings of the International Semantic Web Conference (ISWC)*. 2008.

[11] K. Lua, J. Crowcroft, M. Pias, R. Sharma, *et al.* A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys & Tutorials*, pp. 72–93, 2004.

[12] R. V. van Nieuwpoort, J. Maassen, G. Wrzesinska, R. Hofman, *et al.* Ibis: a flexible and efficient Java based grid programming environment. *Concurrency and Computation: Practice and Experience*, 17(7-8):1079–1107, 2005.

[13] E. Oren, R. Delbru, M. Catasta, R. Cyganiak, *et al.* Sindice.com: A document-oriented lookup index for open linked data. *International Journal of Metadata, Semantics and Ontologies*, 3(1):37–52, 2008.

[14] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, pp. 129–138. 2001.

[15] A. Schlicht and H. Stuckenschmidt. Distributed resolution for ALC. In *Proceedings of the International Workshop on Description Logics*. 2008.

[16] L. Serafini and A. Tamilin. DRAGO: Distributed Reasoning Architecture for the Semantic Web. In *Proceedings of the European Semantic Web Conference (ESWC)*, pp. 361–376. 2005.